


**Web Version of the
Roundabout Simulation Model**

Per Gårder
Bryan Pearce
University of Maine

Final Report

Year 13 (00/01)
Project No. UMEE13-9

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 24, 2002	3. REPORT TYPE AND DATES COVERED Sept 00 – Aug 01		
4. TITLE AND SUBTITLE Web Version of the Roundabout Simulation Model		5. FUNDING NUMBERS DTRS99-G-0001		
6. AUTHOR(S) Per Gärder and Bryan Pearce, University of Maine		PB2002-106717 		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Maine, Department of Civil and Environmental Engineering Orono, ME 04469-5711		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) New England (Region One) UTC Massachusetts Institute of Technology 77 Massachusetts Avenue, Room 1-235 Cambridge, MA 02139		10. SPONSORING / MONITORING Final Report Year 13 (Sept 00- Aug 01), Project No. UMEE13-9		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report describes work done on a roundabout animation program during 2000 and 2001. The roundabout animation program began as an undergraduate class project and was presented in February 1998 in the New England University Transportation Center report "Animation of Traffic through Roundabouts." A second report, "A Roundabout Animation" was presented in June 2000. Undergraduate students were involved in all of these modifications. The program is based on the principle of an autonomous agent. The cars are programmed to speed up, to slow down, and to enter the roundabout based on an acceptable gap length. That is, the gap between themselves and the cars around them. The actual gap is compared to an allowed gap that is based on vehicle speed and assumed driver response time. If necessary, the vehicle speed is adjusted. The cars travel through the roundabout following a randomly assigned path. Traffic flow values may be input into the program manually during initialization. During simulation, the cars enter and exit randomly based on these values. After the simulation, traffic count data and average delay data may be displayed. The latest version has improved input/output modules, is better calibrated against field data, has had some illogical behavior eliminated and is now available on the web through the address www.umeciv.maine.edu/brp/Roundabout .				
14. SUBJECT TERMS			15. NUMBER OF PAGES 22	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	

Technical Report Documentation Page

1. Report No.		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle A Roundabout Animation				5. Report Date June 24, 2002	
				6. Performing Organization Code	
7. Author(s) Per Gårder, Bryan Pearce				8. Performing Organization Report No.	
				10. Work Unit No. (TRAIS)	
9. Performing Organization Name and Address University of Maine Department of Civil and Environmental Engineering Orono, ME 04469-5711				11. Contract or Grant No. DTRS99-G-0001	
				13. Type of Report and Period Covered Final Report Year 13 (Sept 00 - Aug 01)	
12. Sponsoring Agency Name and Address New England (Region One) UTC Massachusetts Institute of Technology 77 Massachusetts Avenue, Room 1-235 Cambridge, MA 02139				14. Sponsoring Agency Code	
				15. Supplementary Notes Supported by a grant from the US Department of Transportation, University Transportation Centers Program	
16. Abstract <p>This report describes work done on a roundabout animation program during 2000 and 2001. The roundabout animation program began as an undergraduate class project and was presented in February 1998 in the New England University Transportation Center report "Animation of Traffic through Roundabouts." A second report, "A Roundabout Animation" was presented in June 2000. Undergraduate students were involved in all of these modifications.</p> <p>The program is based on the principle of an autonomous agent. The cars are programmed to speed up, to slow down, and to enter the roundabout based on an acceptable gap length. That is, the gap between themselves and the cars around them. The actual gap is compared to an allowed gap that is based on vehicle speed and assumed driver response time. If necessary, the vehicle speed is adjusted. The cars travel through the roundabout following a randomly assigned path. Traffic flow values may be input into the program manually during initialization. During simulation, the cars enter and exit randomly based on these values. After the simulation, traffic count data and average delay data may be displayed. The latest version has improved input/output modules, is better calibrated against field data, has had some illogical behavior eliminated and is now available on the web through the address www.umeciv.maine.edu/brp/Roundabout.</p>					
17. Key Words Roundabout, traffic circle, simulation, animation, delay			18. Distribution Statement		
19. Security Classif. (of this report)		20. Security Classif. (of this page)		21. No. of Pages 22	
				22. Price	

Abstract

This report describes work done on a roundabout animation program during 2000 and 2001. The roundabout animation program began as an undergraduate class project and was presented in February 1998 in the New England University Transportation Center report "Animation of Traffic through Roundabouts." A second report, "A Roundabout Animation" was presented in June 2000. Undergraduate students were involved in all of these modifications.

The program is based on the principle of an autonomous agent. The cars are programmed to speed up, to slow down, and to enter the roundabout based on an acceptable gap length. That is, the gap between themselves and the cars around them. The actual gap is compared to an allowed gap that is based on vehicle speed and assumed driver response time. If necessary, the vehicle speed is adjusted. The cars travel through the roundabout following a randomly assigned path. Traffic flow values may be input into the program manually during initialization. During simulation, the cars enter and exit randomly based on these values. After the simulation, traffic count data and average delay data may be displayed. The latest version has improved input/output modules, is better calibrated against field data, has had some illogical behavior eliminated and is now available on the web through the address www.umeciv.maine.edu/brp/Roundabout.

Acknowledgement

We want to thank everybody involved in this project. This includes numerous undergraduate students who during class time suggested methods and procedures for how traffic through roundabouts can be simulated and animated. We particularly want to mention Josh Olund for collecting field data and helping analyze code more in-depth than other students.

We also want to acknowledge the New England University Transportation Center for funding this work. The original funding source is U.S. Department of Transportation.

Web Version of the Roundabout Simulation Model

An Education Project

General

When traffic volumes at an intersection increase to a point where the travel time through it becomes long, or to a point where the intersection becomes unsafe, something should be done to improve the situation. Usually in the United States, the solution is to use a traffic light. However, this does not always solve the delay problem. And, the safety is often improved only marginally. If done right, a roundabout can be a more efficient and safer solution than signalization.^{1 2 3 4}

This report is an update of an ongoing project of the University of Maine Roundabout Model (UMRoM). It describes work done on the roundabout animation program during 2000 and 2001. This animation work started in 1996 as a homework assignment for CIE 115, Computers In Civil Engineering. It was then embraced by one of the students who helped it a few more steps in evolution. The original system has been retained with changes and additions to increase the realism of the modeled traffic. The first report in this series was presented in February 1998 in the New England University Transportation Center report “Animation of Traffic through Roundabouts.” A second report, “A Roundabout Animation,” was presented in June 2000. The program development started with the roundabout as a simple octagon, centered in the window. From there, it has progressed to a circle with entrances and exits. We have now simulated the Gorham Roundabout constructed in 1997⁵ and a couple of yet not built roundabouts. The later versions of UMRoM include a feature to easily allow the setup for any roundabout with six or less entrance approaches and six or less ‘exits’ in any order or combination.

The program is based on the principle of an autonomous agent. When the program is initially set up, each car (the agent) is assigned a set of characteristics that help to define how it should act as it travels through the roundabout. Each vehicle then follows the traffic ‘laws,’ according to those characteristics. The cars are programmed to speed up, to slow down, and to enter the roundabout based on an acceptable gap length. That is, the gap between themselves and the cars around them. The actual gap is compared to an allowed gap that is based on vehicle speed and assumed driver response time. If necessary, the vehicle speed is adjusted. The cars travel through the roundabout following a randomly assigned path. Traffic flow values may be input into the program manually during initialization. Since actual traffic data changes from day to day, and from rush hour to nighttime, we have designed the program to allow the user to easily edit the traf-

-
- 1 Retting, Richard, 1996. Urban Motor Vehicle Crashes and Potential Countermeasures. *Transportation Quarterly* 50/3:19-31.
 - 2 Schoon, C.C., and J. Van Minnen, 1993. Accidents on Roundabouts. R-93-16 SWOV - Stichting Wetenschappelijk Onderzoek Verkeersveiligheid. The Netherlands.
 - 3 Ourston, Leif, 1994. Nonconforming Traffic Circle Becomes Modern Roundabout. Leif ourston and Associates, Santa Barbara, California, 93111.
 - 4 Jorgensen, Else, and N. O. Jorgensen, 1994. Safety of 82 Danish Roundabouts. Report 4 - IVTB, Danish Technical University.
 - 5 Gårder, Per, 1999. Little Falls, Gorham—Reconstruction to a Modern Roundabout, TRRecords No. 1658 Highway Geometric Design and Operational Effect Issues, pp 17 –24.

fic flow data. During simulation, the cars enter and exit randomly based on these values. The program simulates the traffic flow by calculating the vehicle movement in discrete time steps. With each repetition, or timestep, a certain amount of “model time” passes. A variable, ΔT , holds the value of this time step, for example 0.5 seconds. The user can set the simulated time and a data recording time. The traffic simulator shows the vehicles moving along their appropriate paths. The user can also view the “traffic counts” generated by the program. These counts can be converted to vehicles per hour when the simulation is complete), as well as its average time that the cars are in the simulation. Other options allow the user to view all paths, and to have the model operate one step at a time. After the simulation, traffic count data and average delay data may be displayed. And the output of the model includes a continuous real-time animation showing vehicle movements. This includes showing the vehicles’ arrival according to randomized processes based on arrival rates per approach and turn frequencies, the queuing of vehicles on each approach, the entering onto the circulatory lane based on gap acceptance theory, as well as the circulatory and departure processes. Driving speeds are chosen based on desired speeds of that particular autonomous agent combined with queuing theory, i.e., distance to vehicle ahead and its speed. The model’s input and output modules have gradually been improved and the model has been calibrated against real traffic at the Gorham roundabout. Improvements were especially made with respect to the setup procedure for new locations and for differing geometric layouts. The latest version has improved input/output modules, is better calibrated against field data, has had some illogical behavior eliminated and is now available on the web through the address www.umeciv.maine.edu/brp/Roundabout. It is not commercially available in any other way. This was accomplished by involving our undergraduates—in the Civil Engineering computer class CIE115 as well as in the Transportation Engineering class CIE 225. The class “Computers for Civil Engineering,” which has been taught by Bryan Pearce and Will Manion, has had the students work on this module by module. A byproduct of this work has been to stimulate civil engineering majors to specialize in the transportation sector.

Some further details about how this simulation model works are discussed in the following pages.

House Rules

Prior to writing the program, a set of rules were defined that would describe how vehicles behave. First, how could we keep the vehicles from crashing into one another? How is it done in real life situations? Drivers adjust their speed to match that of the car in front of them. Therefore, they will decelerate as soon as they feel they are in danger of hitting that car. Different drivers will do this at different times, depending on how fast they are going. We developed a system such that if a car follows another too closely, then the car in back, or the backcar, will slow.

Merging with traffic at a yield was studied and a system was advanced. We used gap and lag for the model. Lag is called into effect when a car has open road ahead of it and has a car that it might have to yield to. The time it will take for the car on the left to reach the intersection is lag. Gap is a time gap between two cars in the stream of traffic that we are yielding to. On average lag is smaller than gap. When the actual lag or gap is larger than desired the car at the yield will merge.

Next, we needed to decide how the cars would enter the simulation, and where they would enter the simulation. Based on traffic volumes the cars are randomly entered into the simulation. This also depends on the physical space available, that is, a car may not occupy an already occupied space. Once the cars enter, how do they know where to go? In real life, a driver usually knows his or her destination, we randomly assign each car a specific exit, based on traffic volume. This will be explained further in AddCars.

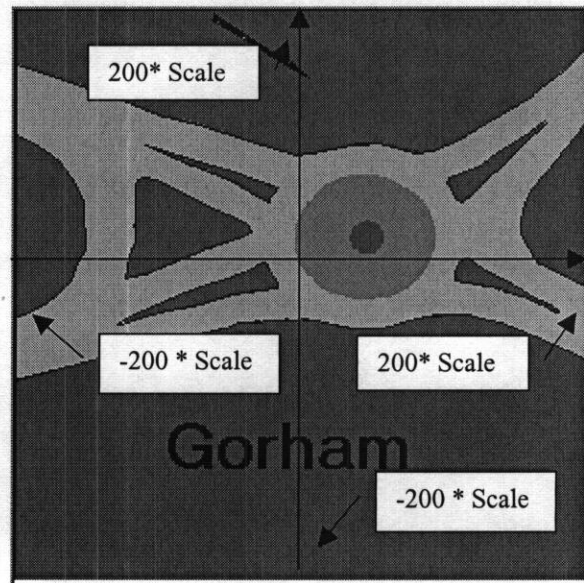


Figure 1 – Coordinate System

Coordinate System

The coordinate system used is a standard Cartesian, ranging from $-200 *$ scale feet to $200 *$ scale feet in both directions (see Figure 1). The scale is defined in the sister program used to prepare the simulation for any given roundabout.

Setup

In the previous version of UMRoM the roundabout and its segments were “hard wired” into the program. That is each value was measured, by hand, and then typed into the code. This process was laborious as well as extremely time consuming. To make the model more useful a sister program was created which allows for rapid setup of any new roundabout (See Figure 2). The program allows the user to load any picture and then click on the picture to place the points that define the segments that the cars follow.

The roundabout setup program allows the user to accomplish many things from telling the program where the roads are to naming the roads. The user may load any standard picture of a roundabout into the program. From here, they are able to pick points over the roads in the picture. To make editing the setup smoother the user is also able to delete or move points. Points are defined by clicking with the mouse. A segment joins two points and has a direction. To make a segment, the user clicks on the first point of the segment and then the second. The first point of a segment would be defined by the simple idea that that point would be the first point in the segment that the car would pass over.

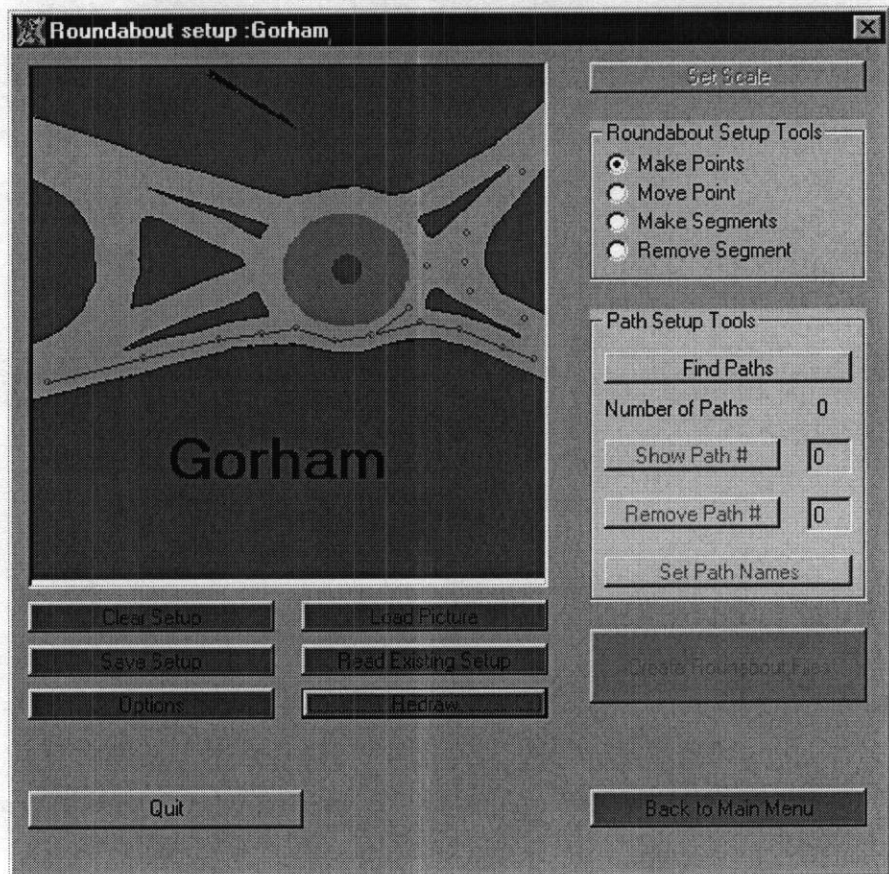


Figure 2 – Setup Form

As with points, the program allows for the removal of segments.

Setups can be saved either in progress or when completed. All setups for the roundabout may be loaded thorough simple save and load dialogue boxes, similar to the ones used to open a file in Microsoft Word. More options are available to the user, such as being able to control the appearance of points, segments, and their labels. One may also make segments and/or points invisible.

Once the segments are placed, the program finds all the possible paths for the traffic flow. The program first finds segments that have a beginning point that does not coincide with the end-point of a different segment. Therefore, that segment must be an entrance into the system. From here, the routine takes that segment and finds another segment that has a starting point that is the same as the first end point. The algorithm will continue to cycle this way from segment to segment and keep track of each path found. When there are two segments branching from an end-point, a routine is used to find which segment is on the right. The path is copied and we continue on to the right segment. The right segment is the segment that then would be the exit of the roundabout (assuming right hand drive).

The process for finding the right segment is done using vector cross products. First, we define the two segments as vectors. Then if we cross the right with the left the resulting vector will be positive. Along with checking for two segments that start from one segment, two segment with a single endpoint (a yield) are also located. The program will again use cross products to find which segment is on the right. This is done to find which of the two segments have to yield.

Once a path is completed, our algorithm will find the next path from the same origin. This is where the copy of the last path is used. Starting from the segment on the left (we took the right last time), we continue looking for "next" segments as before. The program finds all paths until a 360° circuit of the roundabout has been completed. It will discard the redundant path just found move on to the next entrance segment. The cycle continues and stops once all the starting segments have been used.

Now that all the paths have been found the user can then review all of the paths and remove all of the invalid paths. An example of an invalid path is given in Figure 3. A driver must follow the (red) dotted arrow and not the path shown. The user can decide to remove this incorrect path. Now the operator of the program must click on the "set path names" button to name all of the entrances and exits. At this point the user is almost done, all that is left is for the user to set the scale of the drawing by clicking on two points of the map and stating how far apart they are.

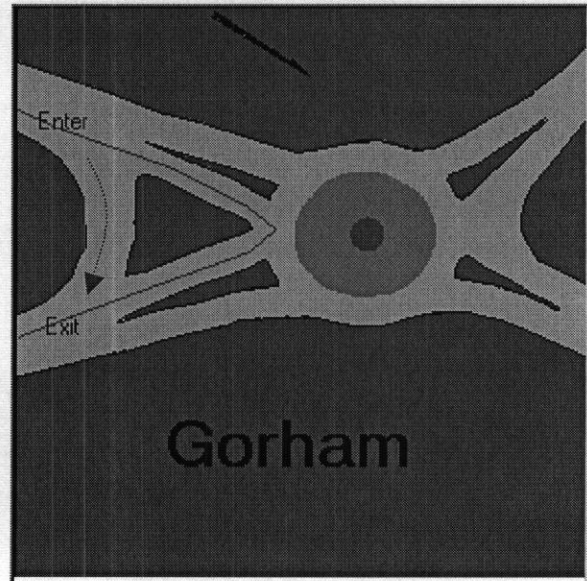


Figure 3 – An Invalid Path

Once the process of setting up the roundabout is done the user clicks the "create roundabout files" button and the information is saved.

The Files

There are four files created by the setup program. They all have the name given by the user but they have different extensions. They are pth, pts, seg, and set. These abbreviations stand for path, points, segments, and segment setup respectively. The path file holds information of how many entrance and exit segments are in the simulation. In addition, this file contains the names of the entrance segments and exit segments. These will be the road and highway names appearing on a real simulation. The last information within this file is combinations of entrances and exits for each path. This data is used to retain the list of segments for each path.

The point file holds the name of the picture that is used for the map and the location of the point used in the simulation. A segment is defined by two "points." These locations are in the Cartesian system as described earlier. The segment file is a list of the ordered numbers of the points and the associated segments. The scale is saved in the segment setup file. Along with scale, the .seg files holds additional information for every segment. The information is stored in the format NextSegL, NextSegR, followed by the

five segments called LeftSegs(1) to LeftSegs(5). These variables will be explained later in this paper.

Code – The Rules in Depth

General Declarations –

Before we get too involved in the explanation of the code, we should describe what variables are being used and why. For those who read this in color, *green italic annotation* is original to the code; blue annotation has been added for clarification.

```
Private Type Point 'this is a variable type defined by us to hold the x and y coordinates of the endpoints of the segments
    x As Single 'the x coordinate
    y As Single 'the y coordinate
End Type
```

```
Private Type segment 'The paths are several linked segments
    carsIn(20) As Integer 'list of cars in each segment – last car first
    totCars As Integer 'number of cars in a segment
    endPt As Integer 'index of segment ending point
    startPt As Integer 'index of beginning point of segment
    length As Single 'length of segment in feet!!!!!!!!!!!!!!!!!!!!!!
    leftSegs(5) As Integer 'when entering the circle look for cars on the five segments to the left
    nextSegL As Integer 'index of next segment continuing in circle - "0" if none
    nextSegR As Integer 'index of next segment leaving circle - "0" if none
End Type
```

```
Private Type Car 'this type defines the properties of the cars
    type As String 'Type of auto. car, bus or truck
    accel As Single 'car's rate of acceleration
    active As Boolean 'if a car is being used or not
    color As Long 'the car's color
    colort As String 'the name of the car's color
    deSpeed As Single 'desired speed or how fast the car "wants" to go
    length As Single 'the car's length
    width As Single 'the car's width
    locationf As Single 'location of car's front in the segment
    locationb As Single 'location of car's back in the segment
    new As Boolean 'if this is true the program will know not to "erase" the car after the first time step
    nextsegf As Integer 'next segment car front is headed for
    nextsegb As Integer 'next segment car back is headed for
    segmentf As Integer 'number of the segment car's front is on
```



```

segmentb As Integer 'number of the segment car's back is on
speed As Single 'actual speed
exit As Integer 'assigned exit segment
begintime As Single 'time the car enters
entrance As Integer 'segment the car enters on
carspeedup As Integer 'for debugging
lag As Single 'lag data for entering circle
gap As Single 'Minimum accepted gap
Follow As Single 'Follow-up value
Tail As Single 'How close a car will get to the one in front of it
yield As Boolean 'Tells if the car is yielding
YieldTime As Single 'The time a car starts to yield, for measuring how
long a car yields
CarsYielded As Integer 'Number times the car has been through the yield
process
End Type

```

```

Private myPts(200) As Point 'array of points
Private mySegs(100) As segment 'array of segments
Private myCars(100) As Car 'array of cars
Private oldFront(100) As Point, oldBack(100) As Point 'these hold the old posi-
tions of the cars
Private numCars As Integer 'number of cars
Private numSegs As Integer 'number of segments
Private deltaT As Single, yieldPt As Single 'the time that passes each timestep,
the point where the cars "look" to enter the circle
Private black As Long 'the color black
Private carFront As Point, carBack As Point 'endpoints of the cars
Dim TimeSteps As Long 'number of timesteps
Dim PutCarInNow(6) As Single 'the probability that a car will enter
Dim counter(6, 6) As Integer 'keeps track of the mean delay time
Dim TimeHolder As Single 'keeps track of when to add a car
Dim frontcar As Integer, backcar As Integer 'these six variables are used to con-
trol speed, the car in front, the car in question
Dim thisseg As Integer, nextseg As Integer 'the segment backcar is on, the seg-
ment it is going to
Dim gap As Single, allowedGap As Single 'the gap between your car (backcar)
and the car in front (frontcar), and minimum gap allowed between them
Dim numpoints As Integer, numsegs As Integer 'number of points and segments
Public roundname As String 'Name of the roundabout files
Private Startn(6, 6) As Way 'An array of entrances and exits
Private startname(6) As String 'The names of the entrances of the Roundabout
Private endname(6) As String 'The names of the exits of the roundabout
Public startnum As Integer 'Number of entrances of the roundabout
Public endnum As Integer 'Number of exits of the roundabout

```

Private Turn(6, 6) As Single
making any given turn.

Private scalenum As Single

Private SpeedDif As Single

Private difference As Single
tested intersection

Private path As String

Private Rtime As Single

Private CurrentTime As Single

'This holds the calculated chance of a vehicle

'Scale of the roundabout files

'Differences in Current car and car yielding to

'Differences in the time it will take to get to con-

'path to where the files are

'For real time calculations

'The current time of the system

These, of course, are not all of the variables used. They are, however, the major ones. The others shall be described as needed.

Initial Code –

When the program starts, a main menu is displayed (See Figure 4). The operator is able to choose whether to simulate a roundabout or prepare a roundabout for simulation. After choosing to simulate, a new form is displayed with only one button active. That is the Input Data button on the Simulation form (See Figure 5). When this button is clicked, an open dialogue box is displayed to allow selection of the roundabout files. Once the roundabout is picked, the program gathers all the in-

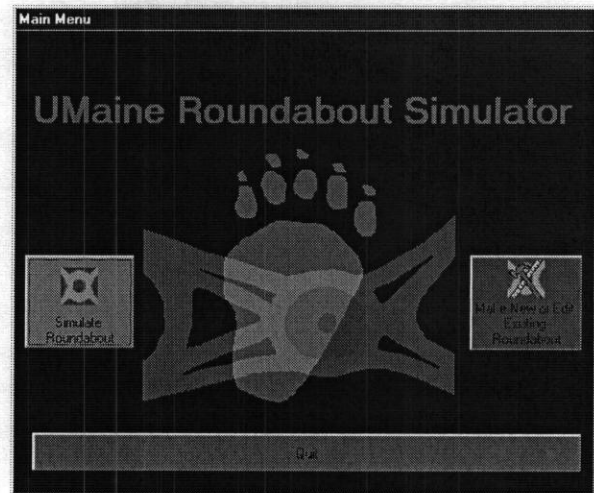


Figure 4 – Main Menu

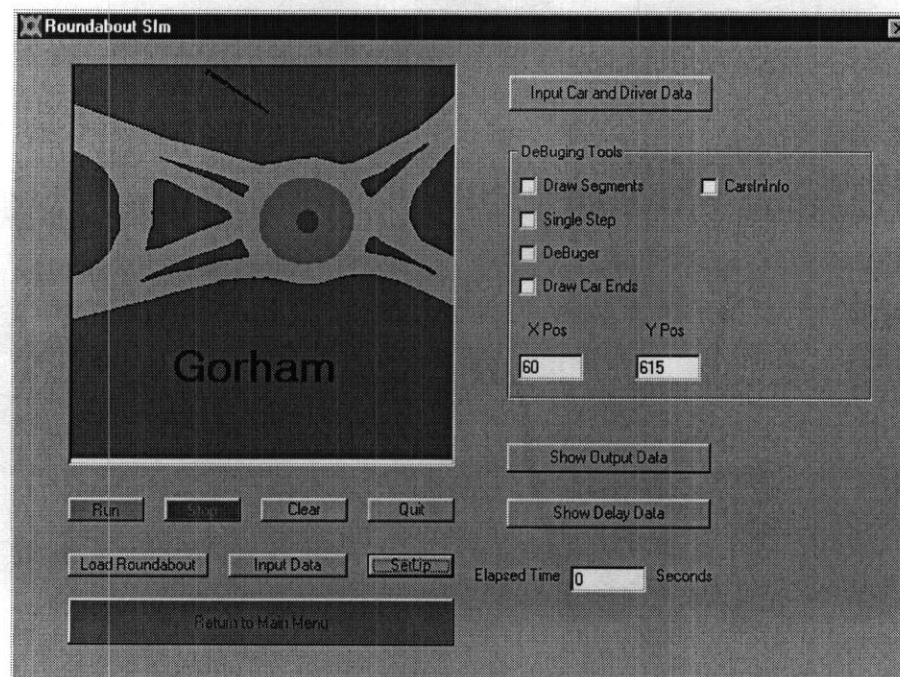


Figure 5 – Simulation Form

formation about the roundabout by reading the appropriate .pth, .pts, .seg, and .set files.

The input data button is now active and when that is clicked a separate form comes up that allows the controlling data for the program to be edited. This new form, frmInput (see Figure 6),

movements of the cars. A "real time" choice has also been added; thus, the time that passes in the simulation is that same as the time that passes in the real world. The default data is entered under the form_load event, that is, when the form loads into memory.

Input Control Data Here

And turn on to

	1	2	3	4
	Hop In North	Frog rd West	Hop In South	Frog rd East
1	Frog rd West	30	37	30
2	Hop In South	0	60	30
3	Frog rd East	30	0	19
4	Hop In North	30	15	0

You can change the data in the grid by clicking on the desired entry and Supplying the new value. Below enter the run length in minutes, and the length of time before data will be recorded in minutes. When done, click O.K.

Run for Minutes

Exclude the First Minutes

Time Steps Minutes ☐ Real Time

Figure 6 – Input Form

Once this is finished, the Setup button becomes active (See Figure 5). The code under (or associated with) this button is fairly straightforward. It initializes variables and properties to be used later, and it calls procedures that setup the points, and segments.

In PointSetup the program opens the file point.txt and reads in the x and y coordinates for each of the points. EndPointSetup “hardwires” the indexes of the points that define each segment. SegSetup sets the length property of each segment, and sets the leftSegs, nextSegL and nextSegR properties. These last two variables help set up the direction in which the traffic will move. They are set up according to the direction of movement. NextSegR will be the segment on the right, if there is a choice. If there is no choice, then nextSegR will have the value of zero and nextSegL will be the next segment. LeftSegs are used to see if there is room to enter the actual circle. If there is a car approaching the circle and there is another car within the circle to the left of the intersection, on one of the LeftSegs, then there is no room to enter and the first car must wait.

PathCalculations –

The Setup button also calls the procedure PathCalculations. This procedure implements the logic that will decide where the cars will enter the circle, when they will enter the circle, and where they will go once they do. PathCalculations reads the data from the MSFlexGrid on frmInput (See Figure 6), and, for each entrance, it calculates the probability that a car would turn in a certain direction. It will also calculate the probability of cars entering each entrance at each timestep.

In computing the probabilities, PathCalculations uses an array that is the number of entrances by the number of exits. For example, if there were 2 entrances and 3 exits then the array would be 2 by 3. The “odds” for a path being taken given the entrance is calculated. The value for 2,1 for example is the simple probability that a car will turn at the first exit. The value of 2,2 is the simple probability of a car making the second turn plus the value for the first. The last value, or 2,3 is the simple probability of a car making the third turn and the value of the second turn, this would be one because this is the last turn.

This will be elaborated on in the explanation of AddCars. The last step in PathCalculations is to originate the PutCarInNow variable for each entrance. The value for these variables is the probability that a car would enter on the appropriate entrance in any timestep.

Timer --

Once the code for the Setup button is finished, the Run button becomes active. Clicking this button enables the Timer, enables the Stop button, and disables itself. The Stop button simply does the opposite of the Run button. However, the Timer becomes the central nervous system of the whole program (See Figure 7). It calls the two major components of

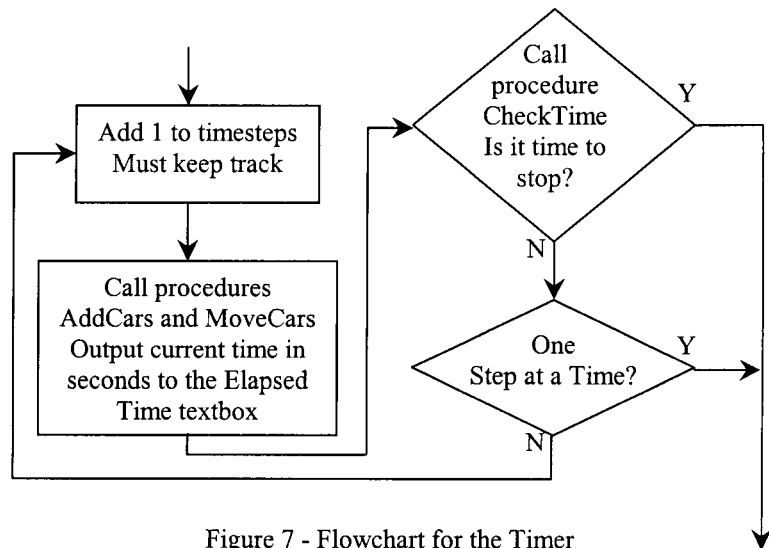


Figure 7 - Flowchart for the Timer

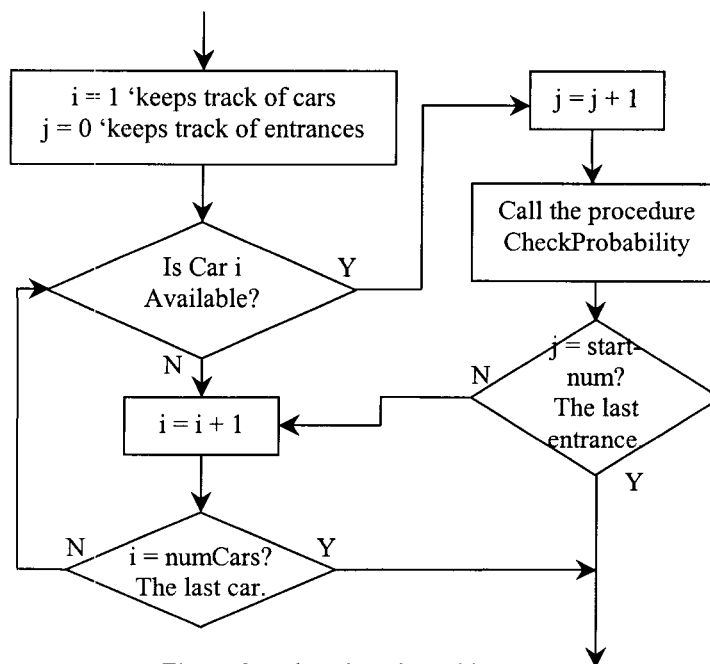


Figure 8 – Flowchart for AddCars

the animation control, AddCars and MoveCars. If the Single Step checkbox is checked then the Timer will turn itself off and turn the Run button back on. This will continue until the box is unchecked. In addition, the Timer calls the procedure CheckTime. This checks to see if it is time to stop the animation and relevant computations, and will do so if necessary.

AddCars –

AddCars searches all the cars being used until it finds the first four that are inactive (See Figure 8). Each car is given a chance to enter one of the four entrances. CheckProbability determines if a car will enter or not (See Figure 9). It does so by first returning a random number. If this number is less than the appropriate PutCarInNow variable, then CheckProbability will let the car enter and the procedure EnterNow will be called. When a vehicle is “allowed in”, the CarSetup routine is called.

CarSetup—

This routine uses the current car in the myCars array and makes sure that the active property is set to true. It also sets the type, dimensions, color and desired speed for each car based on the values given in the traffic statistics window (Figure 10). The operator is al-

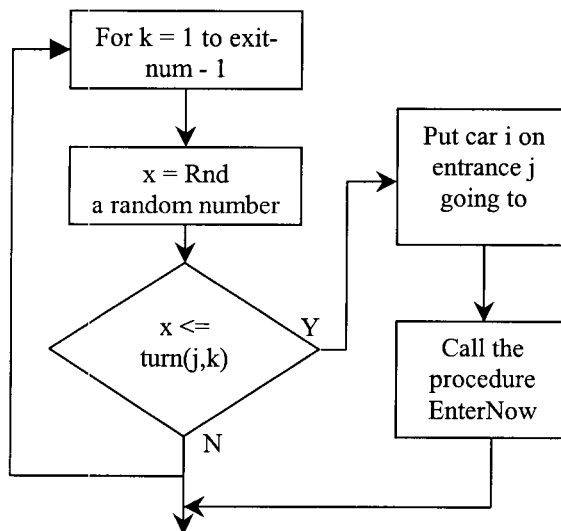


Figure 9 – Flowchart for CheckProbability

lowed to make traffic that consists of any percentage of cars, trucks, and buses. Of course, these various vehicles may have different properties depending on their type. Numbers with a mean and standard deviation are randomly assigned based on a normal distribution. The New Property is set to true so the program knows that this is the first time each car will be drawn.

EnterNow makes the car active and sets its location to the beginning of the entrance segment or behind the last car on the segment if there are cars that are off the screen waiting to enter. Next, the car's destination is made known using the FindExit procedure. This is where the

other variables defined in PathCalculations are used.

FindExit is based on the logic that the sum of all the probabilities of a car taking one of any number of possible paths is one. Remember when finding the values for the array we kept adding the previous variable to the correct probability to get the next. If we use a random number generator to get a number between zero and one, we can use these variables to determine which direction the car will turn. Below is a section of the code with an example.

Traffic Statistics			
Car Statistics		Bus Statistics	Truck Statistics
Car Speed and Acceleration			
Average Car Speed (MPH)	15	Standard Deviation (MPH)	.5
Acceleration (ft/sec ²)	7.7	Standard Deviation (ft/sec ²)	.01
Deceleration (ft/sec ²)	9	Standard Deviation (ft/sec ²)	.01
Average Car Length (Feet)	13	Average Car Width (Feet)	7
Driver Preferences			
Average Lag (seconds)	2.9	Standard Deviation (seconds)	.41
Average Critical gap (seconds)	3.94	Standard Deviation (seconds)	.41
Average Follow-Up (seconds)	2.48	Standard Deviation (seconds)	.24
Tailing Gap (seconds)	2	Standard Deviation (seconds)	.01
Road Statistics			
Yield Point (feet from intersection)	20		
Traffic Mix			
Percent Bus	0 %	Percent Truck	0 %
		Percent Cars	100 %
Done			

Figure 10 – Traffic Statistics

```

j = myCars(i).segment 'the entrance the car is on., let's say 2
x = Rnd 'a random number between 0 and almost 1, let's say 0.754832
For k = 1 To endnum - 1 'endnum in this case will be 3. So k will be 1 then 2.
  If X < Turn(j, k) Then 'first time we are checking turn(2,1) which = .333
    'No .754832 is larger than .333
    'The second time we would be checking turn(2,2) which = .666 again < .754832
    myCars(i).exit = Startn(j, k).ExitSeg
  Exit For
End If

```

```

Next k
If k = endnum Then 'Yes k = 3 and endnum = 3 therefore this is our turn.
    myCars(i).exit = Startn(j, k).ExitSeg 'This sets the segment that the car
                                          'will take for a right turn.
End If

```

The sample car will be turning left. This part has been set up so the exits will be randomly picked, but also they will depend on the vehicles per hour that should follow the paths. X is found using the Rnd function, a random number generator. If it is less than turn(j,k) then it will take the exit for startin(j,k).

EnterNow calls the FindNextSegment procedure after executing FindExit. First, we check to see if there are any segments beyond the segment that the car fronts and backs in question are on (see Figure 11). If there are no segments then the car's nextseg property is set to negative one. Thus, the program will remove the car from the simulation. Next, we want to know if there is a choice of going left or right. If not, the segment's nextSegL property will become the car's nextseg property. However, if there is a choice, we need to know if this is where the car will turn. If the car exits here then nextseg becomes the nextSegR property; otherwise, it is nextSegL. Interested readers may want to look at the code:

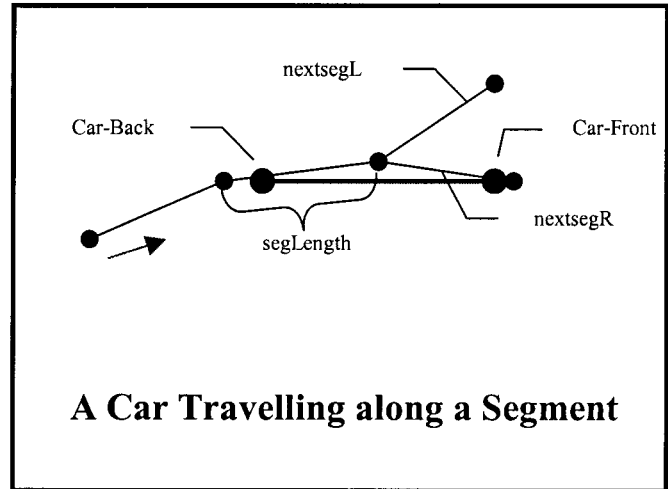


Figure 11 – Car-Fronts and Backs

```

segin = myCars(i).segmentf 'segment the front of car i is in
If segin = -1 Then 'The front of our car is out of the simulation and ignore it.
    MyCars(I).nextsegf = -1
ElseIf mySegs(segin).nextSegL = 0 Then 'We are on the exit ramp
    myCars(i).nextsegf = -1
ElseIf mySegs(segin).nextSegR = 0 Then 'There is only one segment in front of
this segment
    myCars(i).nextsegf = mySegs(segin).nextSegL
ElseIf myCars(i).exit = mySegs(segin).nextSegR Then 'This is the exit the car is
looking for and will take the right turn.
    myCars(i).nextsegf = mySegs(segin).nextSegR
Else 'This is not the car's exit and will stay in the roundabout.
    myCars(i).nextsegf = mySegs(segin).nextSegL
End If

```

This routine is repeated for the back point of the car as well. This procedure has worked very well, and has been only slightly modified since the spring course in 1996.

The most important of these was in changing how the cars decided to turn off the circle. In the original homework program, the decision was made by probability within FindNextSegment. Now the exit is already known, a priori, all we need to do is compare nextSegR with the exit.

Once we have found the next segment for which we are heading, we need to update the carsIn array. The carsIn array is set up to hold 50 numbers but we have to have slots for both the fronts of cars and the backs of cars. So, the carsIn array will hold 25 cars. The odd carsIn (carsIn(1), carsIn(3), carsIn(5)...) hold the values of the back of the car that is in the spot. A zero means no car. The even carsIn (carsIn(2), carsIn(4), carsIn(6)...) hold the values of the front of a car that is in the spot. When updating the array, we move all of the cars in the array to the next two highest slot in the array for example (carsIn(1) goes to carsIn(3) and carsIn(2) goes to carsIn(4)) and put our car in slot one or two.

We now know where the car has entered and where it will exit, and can update the grid on frmVPH (see Figure 12). EnterNow calls the procedure UpdateOutput to accomplish this task. Of course, UpdateOutput will do nothing unless the elapsed time has reached the time at which data starts recording. When it is time to record data, the procedure enters another nested case statement (this one relies on the car's entrance and exit) to determine which cell in the grid to increment by one.

Before the car moves, we must adjust the car's speed so it will not be likely to crash into the car in front of it. We do this by calling the procedure AdjustSpeeds. (The crash rate at a modern roundabout is typically around one per 2 million entering vehicles.⁶ We disregard this small probability in our simulation, and assume that cars do not collide when estimating average delays.

	Hop In	Frog rd	Hop In	Frog rd	Entr. Sum
Frog rd	28.0	28.0	24.0	.0	80.0
Hop In	.0	64.0	12.0	56.0	132.0
Frog rd	28.0	.0	28.0	16.0	72.0
Hop In	16.0	8.0	.0	16.0	40.0

Figure 12 – Vehicles Per Hour Display

The last two steps EnterNow performs are to set the car's begintime property and the car's entrance property. These will be used later in the procedure FindDelayTime.

AdjustSpeeds –

In AdjustSpeeds, we set values to the variables: accel, taken from the acceleration property of the given car; backcar, car i or the car being controlled (conceptually the car

⁶ Gårder, Per, 1998. Little Falls, Gorham—A Modern Roundabout, Maine Department of Transportation, Bureau of Planning, Research & Community Services, Transportation Research Division, Final Report, Technical Report 96-2b.

we are in which we are riding); thisseg, the segment backcar is on; and nextseg, the segment to where backcar is heading. Typical speed and acceleration values have been obtained within this project, and are also presented in separate publications.^{7 8} The logic for AdjustSpeeds is given in the flow chart labeled Figure 13.

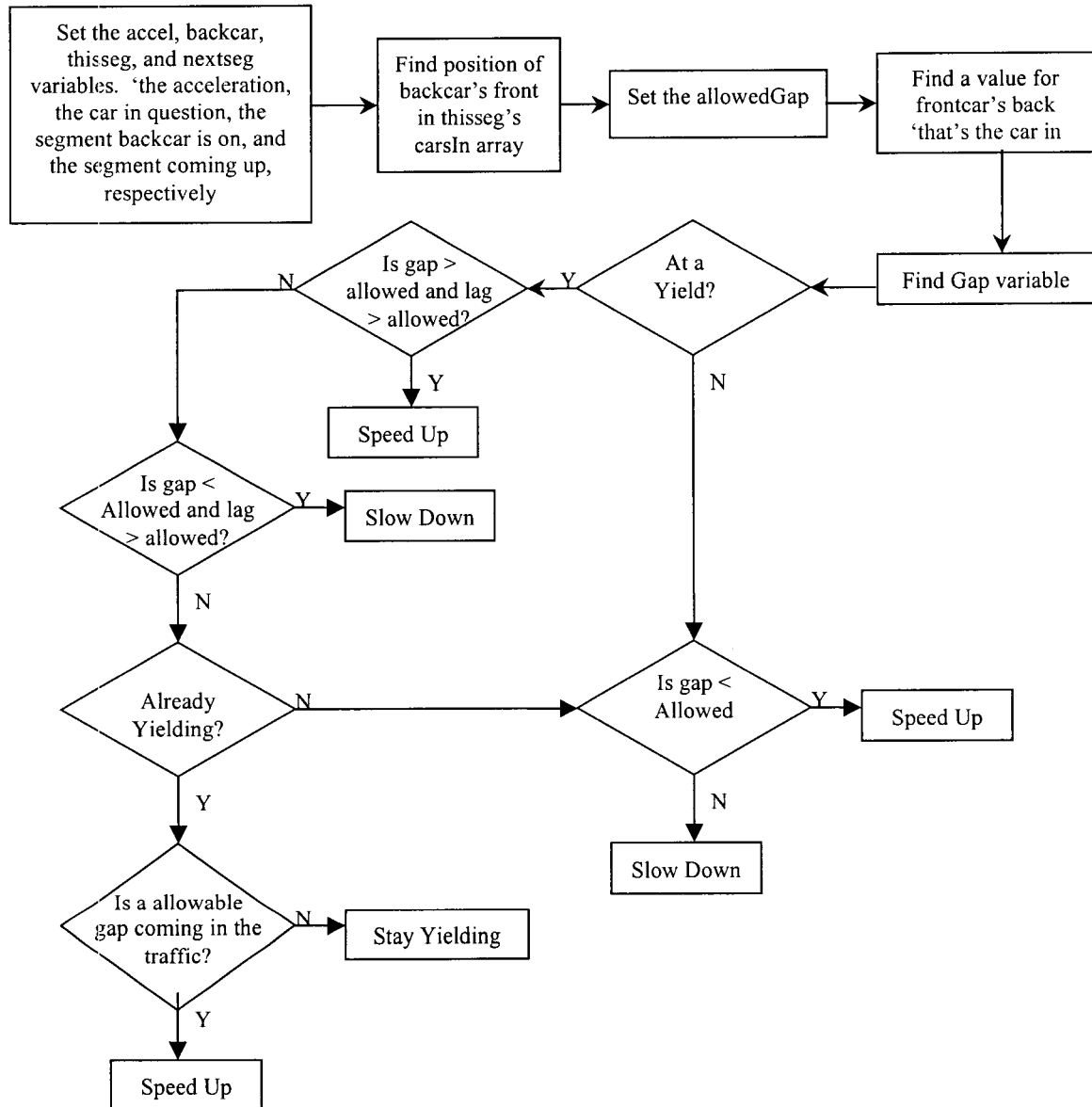


Figure 13 – Flowchart for AdjustSpeeds

At this point, AdjustSpeeds enters a “do loop” in order to find a value for *j*. This is the index of the place of the front of the backcar in the carsIn array for thisseg. *j* becomes

⁷ Gårder, Per, 1998. Little Falls, Gorham—A Modern Roundabout, Maine Department of Transportation, Bureau of Planning, Research & Community Services, Transportation Research Division, Final Report, Technical Report 96-2b.

⁸ Modern Roundabout Practice in the United States. A Synthesis of Highway Practice. NCHRP Synthesis 264, TRB, Washington D.C., 1998.

useful when trying to find the value of frontcar, the car in front of backcar. Here we set the values of lookLeftSeg and lookLeftCar. They are used if the car is approaching an intersection. What happens then will be explained later.

The routine sets the speedup variable to a number from 1 to 4. A 1 means the car can speed up; a 2 means that the car has to slow down; a 3 means that the car yields to another car in the roundabout; finally a 4 means that the car is speeding up from a stopped position. Now, we find the value of allowedGap, using the equation: $\text{allowedGap} = \text{myCars}(i).\text{Tail} * \text{myCars}(\text{backcar}).\text{speed} + 2$. The $\text{myCars}(i).\text{Tail}$ is the value taken from car statistics window. $\text{myCars}(\text{backcar}).\text{speed}$ is backcar's speed. The two is added because if the speed were 0 the allowed gap would be 0 and that would cause problems.

Once these variables are defined, we can find the value for the back of the frontcar. First, we check thisseg. Because of the way the carsIn array is set up, we can see if frontcar is on thisseg by adding one to j and then checking that spot in the array for a number other than zero. A non-zero number will indicate the back of the frontcar, and Gap's value will be the difference of frontcar's back location and backcar's front location. If there is no car in front of backcar on thisseg, we look to see if nextseg is negative one, meaning that the car will be leaving the simulation. If this is the case nothing needs to be done and we can move on. However, if this is not the case, we need to find out how far ahead the next car is. We do this by calling the aptly named procedure FindFrontCar.

FindFrontCar uses the same variables we have already defined in AdjustSpeeds. It first initializes the Gap variable with the length of thisseg that backcar has not yet traveled. Then it uses a Do loop to find frontcar. Let us look at the code:

```
gap = mySegs(thisseg2).length - myCars(backcar).locationf 'the remaining length
of thisseg
Do 'the program will loop back to here
frontcar = mySegs(nextseg2).CarsIn(1) 'the rear of the last car on nextseg
frontcarfront = mySegs(nextseg2).CarsIn(2) 'the front of the last car on nextseg

If frontcar <> 0 Then 'is there a car on nextseg?
    gap = gap + myCars(frontcar).locationb 'add the length of nextseg trav-
eled by frontcar if there is one.
    Exit Do
Else
    gap = gap + mySegs(nextseg2).length 'add the entire length of nextseg if
there is not.
End If
thisseg2 = nextseg2 'if frontcar wasn't found we need to find the next nextseg
If mySegs(thisseg2).nextSegR = 0 Then 'this part is just like FindNextSegment
    nextseg2 = mySegs(thisseg2).nextSegL
ElseIf myCars(backcar).exit = mySegs(thisseg2).nextSegR Then
    nextseg2 = mySegs(thisseg2).nextSegR
Else
    nextseg2 = mySegs(thisseg2).nextSegL
End If
```

```

    If nextseg2 = 0 Then gap = 4 * allowedGap + gap 'to make sure that the gap will
    be large and there won't be abnormal slowing down right before exiting the simulation.
    Loop Until gap > allowedGap Or thisseg2 >= 99 'keep looking until it doesn't
    matter

```

The second to last line was added because the cars were slowing prematurely, just before they got on the last segment before leaving the simulation. As they came close to the beginning of the last segment, the gap would only be a small amount plus the length of the last segment. Thus, Gap would be smaller than allowedGap, and the car would slow. Typical values of critical gaps were observed and researched through studies of literature within this project and integrated into the code.

FindFrontCar was created to find an accurate value for Gap. With this value, we can now adjust backcar's speed if necessary. If Gap is less than the allowedGap, we need to slow backcar and speedup is set to 2. We also must check to see if there is a car to the left at a yield. If there is a car to the left we find how far away it is. If the car is back further than the acceptable lag then we will go faster if we can, speedup = 1. Lag is the time it takes for a car that one would have to yield to, to reach the intersection. When the car to the left is closer than the acceptable lag we will slow to a stop and yield to the other car, speedup = 3. When we are yielding we start looking for an acceptable gap so we can drive into the roundabout, speedup = 4.

Once the speedup is defined, we act accordingly. If speeding = 1, we will accelerate. To do this we add the product of accel, the acceleration in ft/sec^2 and the time that has passed since last we checked, deltaT to backcar's speed. Next, we check backcar's speed. if more than the desired speed, we set the speed to the desired speed. When speedup is 2, the car must slow because it is getting too close to the car in front of it. This is done by subtracting the product of modaccel, the acceleration in ft/sec^2 and the time that has passed since last we checked deltaT from backcar's speed. Modaccel is the modified acceleration of the car. This takes in account how close the two cars are and the difference in their speeds. Therefore, when one car is "coming up fast" to a slow car it will brake harder. Again, we check the new speed, if it is less than 0 it is set to 0. A speedup of 3 means that the car has to yield. This is the same processes of slowing down but the yield variable is set to true, to know that the car is yielding. On a speedup of a 4, the driver give it a little higher acceleration so the car will join traffic smoothly.

MoveCars –

The procedure MoveCars has been split into two sub-procedures: SwitchSegments, which moves the cars along, calls adjust speeds and switches the segments the cars are on when necessary; and DrawCars, which draws and erases the cars as they move around the traffic circle.

SwitchSegments loops through all the "active" cars (the cars with their active property set to true). It calls AdjustSpeeds for every one, and advances the cars along the segments. It does the latter using the equation: $\text{myCars}(i).\text{locationf} = \text{myCars}(i).\text{locationf} + \text{myCars}(i).\text{speed} * \text{deltaT}$, and: $\text{myCars}(i).\text{locationb} = \text{myCars}(i).\text{locationb} + \text{myCars}(i).\text{speed} * \text{deltaT}$. This moves both the front and rear of the car the distance the car would have traveled in one timestep. Therefore, when we multiply the car's speed with the elapsed time we get the distance the car has actually gone.

If either part of the car's location becomes larger than the length of the segment it is on, then the car has moved onto the next segment and must be treated accordingly. First, the carsIn array for thisseg must be updated. When that is done, we check to see if the car is leaving the simulation, in which case nextseg equals negative one. If it is, then the car is made inactive, New Property is set to be true, and it is erased so that it does not leave a "blip" on the screen. We also call the procedure FindDelayTime at this point. This uses a nested case statement to find the proper cell in the MSFlexGrid control. Once found, it will update the average delay (see Figure 14). If the car is not leaving the simulation, then we must update the carsIn array of nextseg. After that, we call FindNextSegment to find the new nextseg, and correct the car's location so it will fit its new segment.

The animation is the apparent motion of our drawings. In actuality the drawings are not moving.

In the traffic circle program, the cars, are being drawn and erased and drawn again in a different place, creating the illusion of motion. Initially, the cars were just round dots. When we extended them into lines, we had trouble orienting them. The current method of drawing the cars is by drawing a line from the front of the car to the back of the car. This helps to make the vehicle appear to be making smooth turns.

DrawCars loops through all the "active cars", and will find their new x and y coordinates, measured at the two ends of the car. It does this by finding the endpoints of the segment the car is on and interpolating the car's coordinates using the coordinates of the endpoints and the car's location. It will then find the angle of the segment, theta, and the car's endpoints as described above. After setting the draw width to the width of the car, it will draw the car using Visual Basic's line method. If the car's new property is not set to true then DrawCars will erase the old drawing of the car at its old position. If it were set to true then DrawCars would do nothing except set the new property to false, because there is no old drawing to erase. Lastly, DrawCars will store the coordinates of the car's endpoints so it can erase it during the next time step.

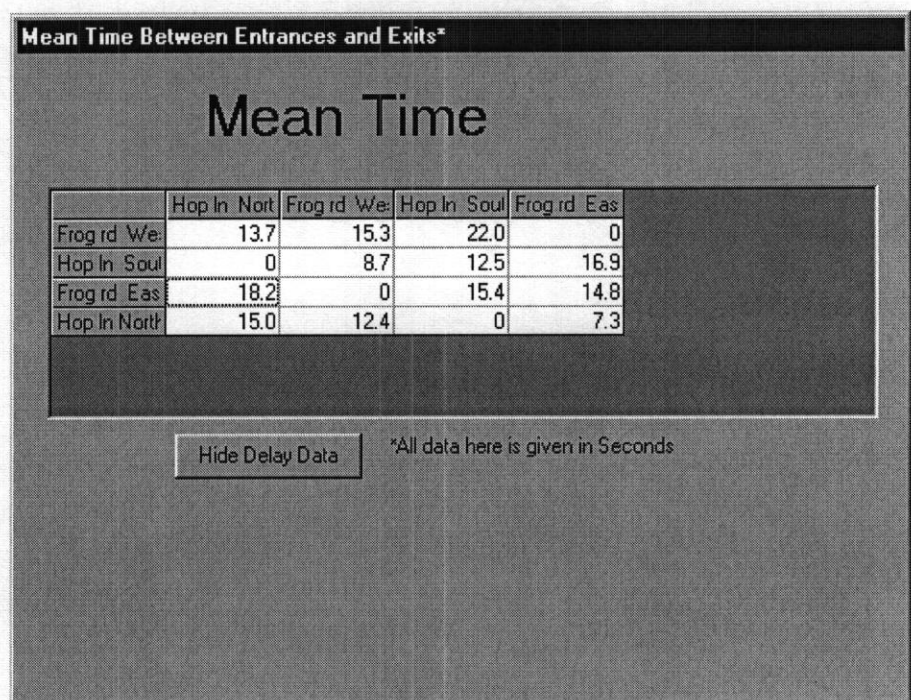


Figure 14 – Mean Time of Traffic